SpikeTemp: an Enhanced Rank-Order Based Learning Approach for Spiking Neural Networks with Adaptive Structure

Jinling Wang, Ammar Belatreche, *Member, IEEE*, Liam Maguire and T.M. McGinnity

***Abstract*— This paper presents an enhanced rank-order based learning algorithm, called SpikeTemp, for Spiking Neural Networks (SNNs) with a dynamically adaptive structure. The trained feed-forward SNN consists of two layers of spiking neurons: an encoding layer which temporally encodes real valued features into spatio-temporal spike patterns, and an output layer of dynamically grown neurons which perform spatio-temporal classification. Both Gaussian receptive fields and square cosine population encoding schemes are employed to encode real-valued features into spatio-temporal spike patterns. Unlike the rank-order based learning approach, SpikeTemp uses the precise times of the incoming spikes for adjusting the synaptic weights such that early spikes result in a large weight change and late spikes lead to a smaller weight change. This removes the need to rank all the incoming spikes and thus reduces the computational cost of SpikeTemp. The proposed SpikeTemp algorithm is demonstrated on several benchmark datasets and on an image recognition task. The results show that SpikeTemp can achieve better classification performance and is much faster than the existing rank-order based learning approach. In addition, the number of output neurons is much smaller when the square cosine encoding scheme is employed. Furthermore, SpikeTemp is benchmarked against a selection of existing machine learning algorithms and the results demonstrate the ability of SpikeTemp to classify different datasets after just one presentation of the training samples with comparable classification performance.**

***Index Terms*—adaptive spiking neural networks, clustering, classification, online learning, spiking neurons, supervised learning.**

I. Introduction

An artificial neural network (ANN) is a biologically inspired information processing paradigm which mimics the way the brain acquires and processes sensory information [1]. ANNs have been researched extensively and have successfully been used in a wide range of applications. One of the fundamental issues in neuroscience is the problem of neuronal coding; despite significant progress having been made in understanding the dynamics of biological neurons, there is no definitive understanding of what code is used by neurons to represent and transmit information in the brain. It has been verified that thousands of spikes are emitted per millisecond in a very small area of the cortex, and that information is transmitted and processed efficiently in the brain [2]. The main motivation behind the study of these biologically plausible neuron models is to further our understanding of how they communicate, how computation is carried out in the brain [3-8], and also to understand brain function and dysfunction (neurodegenerative diseases). The ultimate goal from a computing perspective is to exploit such knowledge in devising novel sophisticated intelligent computational systems. To date, a number of supervised and unsupervised learning methods [9-35] have been developed for SNNs; a review of some of these learning rules can be found in [36][37]. With a few exceptions [33-35] these efforts have found limited success in applying spiking neural networks to solving real-world problems due to the lack of efficient and scalable learning algorithms. Most of the existing learning algorithms require retraining if used in a changing environment and fail to scale. Therefore, further development is still needed to devise efficient and scalable online learning mechanisms for spiking neural networks (SNNs) in order to increase their applicability in solving real world problems.

SpikeProp represents an adaptation of the classical backpropagation algorithm, and was the first supervised learning algorithm developed for SNNs [9]. Its performance on several benchmark datasets, including non-linearly separable classification problems, demonstrated that SNNs with temporal coding can achieve comparable results to classical rate-coded networks [9] [10]. However, there were several issues this algorithm needed to address, such as slow convergence especially for large datasets and the problem of non-firing (silent) neurons. Subsequently, several methods have been developed to improve SpikeProp [11-15]. These gradient based algorithms are computationally powerful but are often regarded as non-biologically plausible because they require a non-local spread of error signals from one synapse to another. Besides, they are slow if used in an online setting, and getting stuck in local minima is another well known problem for gradient-based approaches. Belatreche et al. [16] proposed a derivative-free supervised learning algorithm

This manuscript was submitted on March, 12, 2015; revised on July 18, 2015.

The authors are with the Intelligent Systems Research Centre, School of Computing and Intelligent Systems, University of Ulster, Londonderry, Northern Ireland BT48 7JL, United Kingdom. (e-mail: wang-j1@email.ulster.ac.uk; a.belatreche@ulster.ac.uk; lp.maguire@ulster.ac.uk; tm.mcginnity@ulster.ac.uk).

T.M. McGinnity is also with the School of Science and Technology, Nottingham Trent University, UK.

where an evolutionary strategy (ES) was used to minimise the error between the output firing times and the corresponding desired firing times. This algorithm achieved a better performance than SpikeProp. However, since the algorithm was an ES-based iterative process, the training procedure was extremely time-consuming and is not suitable for online learning.

Spike-Timing Dependent Plasticity (STDP) and Hebbian learning are biologically plausible learning rules [17]. Like Hebbian learning, STDP is unsupervised, which is applied locally to a synapse linking pre-synaptic and post-synaptic neurons. The synaptic plasticity depends on the relative timings of pre-synaptic and post-synaptic spikes. STDP-based learning has been investigated in supervised learning [18-23], in unsupervised learning [24][25], in reinforcement learning [26] and in associative memory [27] [28].

Legenstein et al. [18] presented a Supervised-Hebbian learning method (SHL) which forces the post-synaptic neuron to fire at specific desired times using an extra 'teaching' input signal. The algorithm was able to implement different transformations between input spike trains and output spike trains quite well, however it was reported that convergence cannot be guaranteed in a general case and that synaptic parameters continue to change even if the neurons fires exactly at the desired times.

ReSuMe, another supervised learning method, developed by Ponulak [19][20][21], integrated the idea of learning-windows with remote supervision. It was shown that the desired temporal sequences of spikes can efficiently be learnt after projection of the input data on a Liquid State Machine (LSM) network. The authors claimed the method is suitable for online processing. However, the network structure used in this method is fixed and does not adapt to incoming stimuli. In addition, the desired precise output spike timing is crucial to ReSuMe learning.

Another supervised learning, called the Tempotron, was proposed by Gutig and Sompolinsky [22]. It updates the weights of a neuron using an error function which is based on the difference between the maximum membrane potential and the threshold of this neuron so that it fires or remains silent depending on whether the presented spiking inputs belong to one class or another, respectively. It was, however, reported by Florian [23] that this learning rule is in fact equivalent to a special case of ReSuMe under certain conditions.

Masquelier et al. [24] presented an unsupervised STDP-based learning approach, in which a single neuron uses STDP learning process to successfully detect and learn a repeating arbitrary spatiotemporal spike pattern that is hidden in equally dense distracter spike trains. This approach was later extended to multiple repeating patterns and multiple STDP neurons [25] where competition between neurons is achieved through the use of lateral inhibitory connections.

Legenstein et al. [26] presented another unsupervised learning rule based on reward modulated STDP where complex firing patterns of presynaptic neurons can be distinguished with no need for a supervisor to instruct the neuron when it should fire. This method is sensitive to local fluctuations of the membrane voltage rather than the peak value of membrane voltage as in the Tempotron learning [22].

Scarpetta and Giacco [28] use an STDP-based learning process to study the collective dynamics of a Leaky Integrate and Fire network, so that the resulted network can work as associative memory, in which precise relative phase relationship of spikes among neurons are stored then recalled. This model stores not only the order of activation in a sequence, but the precise relative times between spikes in a phase-coded pattern. After changing the excitability parameters of the network, different regimes are observed and discussed.

It is important to note though that these STDP-based learning methods (both supervised and unsupervised) are batch training methods with fixed network structures. That is, their networks do not evolve during learning, hence they do not adapt to incoming stimuli, which make them in current form unsuitable for online learning.

The work of Thorpe et al. [29] has shown that the visual system is capable of processing complex natural scenes in a timescale of 100-150ms. A consideration of the fact that such a task is completed so quickly despite passing through many areas of the brain which is composed of billions of neurons led to the suggestion that the first spike should contain most of the information; this is reflected in the time-to-first spike encoding scheme. The authors [30-32] therefore proposed an offline rank-order based learning approach for a feedforward SNN of integrate-and-fire neurons, which uses only one spike per neuron and can classify faces successfully. However, two issues were highlighted in [33]; first, since the weight change is determined by a modular factor and the number of training samples, then the number of training samples needs be known in advance; and second, the trained network is selective to the average pattern, so it is not suitable for online learning.

All of the above-mentioned approaches use an SNN with a fixed structure, where the sizes of the hidden and output layers must be specified a priori, and are trained in an offline batch mode. Therefore, these approaches can only be applied if the number of classes or clusters is known up front. In addition, these approaches cannot be applied to problems where data is continuously changing as they will need to retrain both the old and new data samples. However, biological neural networks are known for their ability to learn continuously and incrementally which account for their continuous adaptation to changing non-stationary environments. Therefore, to allow a spiking neural network to interact with a continuously changing environment, it is necessary that both its structure and weights dynamically adapt to new data. Also, catastrophic interference/forgetting should be avoided when new information is learned.

Wysoski et al. [33] selected the offline learning procedure in [30]-[32] with a fixed structure and adapted it to online learning with an adaptive network structure. The model presented in [33] consists of a four layer hierarchical neural network of two-dimensional integrate-and-fire neuronal maps. The proposed procedure can perform learning in an online mode through synaptic plasticity and adaptive network

structure. The training procedure was applied to a publicly available face recognition dataset, and the performance obtained was comparable to the optimised off-line method. In [33], the facial images are firstly preprocessed, the boundaries of the region of interest (ROI) are chosen manually between the inter-ocular distance and the distance between the eyes, and then the ROI is normalized to a size of 20 x 30 pixels; after an image is pre-processed to the size 20*30 pixels in greyscale, it is used as input to the SNN. In real time applications, many data samples are 1D feature vectors, so in [34], Gaussian population encoding is used to encode every input feature into a set of spike times with a population of neurons such that each neuron can spike only once and then a rank order coding learning method is employed for the learning. The learning method used to train the weights in [33][34] is based on the rank order of the incoming spikes arrival. However, in these networks [33][34], several issues are highlighted: (a) The learning method used to train the weights is based on the order of the incoming spiking arrival. The precise timing information is thrown away despite the fact that the precise times not only carry the rank order information, but also how different they are [39]; (b) due to the time spent on the calculation of the rank order, the simulation time of the network is slow for large datasets and networks; (c) In [33], it has been shown that SNN can be used to extract face images features, the network presented is suitable for 2D inputs; however, in real world application, many inputs are represented by a 1D feature vector and the pre-processing of a face image in [33] is time-consuming for an online system.

This paper presents an enhanced rank-order based learning method, called SpikeTemp, for spiking neural networks with an adaptive structure where, unlike the existing rank-order based learning approach [33][34], the precise times of incoming spikes are used to determine the required change in synaptic weights. The proposed approach employs a two-layer feed-forward spiking network with a layer of encoding neurons and a layer of output neurons. It is suitable for inputs represented by a 1D feature vector. It is more appropriate for online systems. SpikeTemp calculates the weight changes between the encoding layer and the output layer based on the precise times of the incoming spikes such that the amount of weight change decreases exponentially with later spike times, i.e. early spikes result in a larger weight change and late spikes lead to a smaller weight change. This removes the need to explicitly rank order incoming spikes. As a result, SpikeTemp is computationally efficient and is more applicable for a wide range of datasets. Furthermore, in addition to the Gaussian receptive field population encoding scheme, the square cosine population encoding is also employed in SpikeTemp for temporally encoding the input features into spatio-temporal spike patterns.

The remainder of this paper is structured as follows: Section 2 describes the employed neural model, the two temporal encoding schemes and presents the SNN structure design. Section 3 presents network structure adaptation and learning procedure. Section 4 presents experimental results for training SNNs, using both encoding schemes, on selected benchmark datasets from the UCI Machine Learning Repository. The results obtained are compared with those obtained from the rank order approach as well as standard classical machine learning methods. Section 5 describes the application of SpikeTemp to a visual pattern recognition task and Section 6 provides an analysis and discussion of various parameters effect on the learning performance. Finally section 7 concludes the paper and outlines future work.

II. Neural model, information encoding schemes and network structure

A. Spiking Neural Model

Neuronal models with varying degrees of computational complexity have been developed and reported in the literature [2][38]. For the proposed SpikeTemp algorithm, it was considered important to choose a tractable yet biologically relevant neuron model in order to reduce the computational complexity of the spiking neural network which is critical for online learning. Balancing biological plausibility and tractability, SpikeTemp employs simple integrate-and-fire (IF) neurons in output layer that are also employed in related work [30-32]. The detailed dynamics of this model were analysed and explained in [30]. After a spike is generated in the output layer, the simulation for the current input sample is terminated and the PSP of firing output neuron is reset and the neuron remains silent. The postsynaptic potential (PSP) of an output neuron i at time t relies on the spike times received from neurons in the encoding layer and can be described as:

$$PSP(i,t) = \sum_{\epsilon} - \qquad (1)$$

Where $j \in [1, N]$ represents the j^{th} incoming connection, and N is the total number of incoming connections between the encoding layer and the output neuron i; t_j represents the precise spiking time of the j^{th} encoding neuron; τ is a time constant and determines the range for which each synaptic strengthening occurs; W_{ji} is the synaptic weight associated with the synaptic connection between output neuron i and encoding neuron j. If $PSP(i, t)$ is greater than the firing threshold, $PSP_{th}(i)$, of neuron i , then an output spike isproduced at neuron i in the output layer, and the simulation for the current input sample is terminated.

B. Information encoding

The Gaussian Receptive Field population encoding scheme, proposed by Bohte et al. [10], can be used to encode continuous input variables into spike times. The input can be distributed over several neurons with overlapping and graded sensitivity profiles, e.g., Gaussian activation functions. In [10] there is detailed description of how to set the centre and width of a neuron. In this work the parameter β is taken as 1.5 as used in [10]. Each encoding neuron fires only once during the time coding interval $[0, T_{ref}]$; T_{ref} = 9ms was employed in this work (this value is chosen arbitrarily). As a result, each input

sample is translated into a spatio-temporal spike pattern. An example is shown in Fig. 1 where a real-valued feature of 5.1 (illustrated by a vertical dashed red line) is converted into spike times using eight Gaussian receptive fields. Response values (*y*) can be obtained from points where the vertical dashed line at 5.1 intersects the Gaussian curves and the resulting eight response values are 0.0228, 0.4578, 0.9692, 0.2163, 0.0051, 0, 0 and 0 respectively (note that all values lie between 0 and 1). These values are then mapped linearly to spike times (see (2)) such as the highest response value of 1 is associated with spiking time t=0ms (i.e. early firing time) and the lowest response value of 0 is associated with spiking time t=9ms ((i.e. late firing time).

(2)

If the resulting spike time is equal to 9ms, this neuron is treated as 'silent', and is represented by a value of '-1'. Also, the resulting spiking times are rounded to two decimal values (the nearest time step) in ms. For example, the resulting spiking times which encode the single real-valued feature of 5.1 using 8 receptive fields input neurons are therefore represented by the following series of spiking times: 8.79ms, 4.88ms, 0.28ms, 7.05ms, 8.95, -1(silent), -1(silent) and -1(silent).

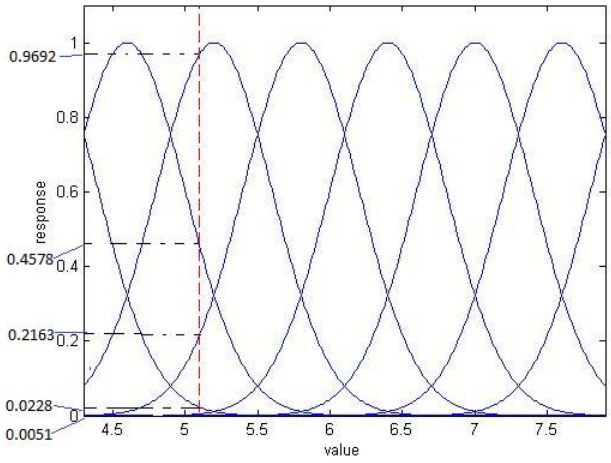

Fig. 1. Encoding of a real valued feature of 5.1 using 8 Gaussian receptive fields neurons.

Wu et al. [14] proposed a square cosine encoding method that was used to code continuous input variables into spike times to improve the precision of the encoded data. An example is shown in Fig. 2 where a feature value of 5.1 is converted into spike times using eight Square Cosine encoders. Spike times can again be obtained from points where the vertical dashed line intersects the square cosine curves; the resulting encoding values are 0.9096, 0.9924, 0.7868, 0.4132, 0.0904, 0.0076, 0.2132 and 0.5868, respectively for value 5.1 (note all values lie between 0 and 1). These values are again converted linearly into spike times by associating the highest response value 1 with spiking time t=0ms and the lowest response value 0 with spiking time t=9ms (see (2)). The resulting spiking times are rounded to two decimal values, so the converted spiking times for the single input value 5.1 using 8 input neurons are 0.81, 0.07, 1.92, 5.28, 8.19, 8.93, 7.08 and 3.72.

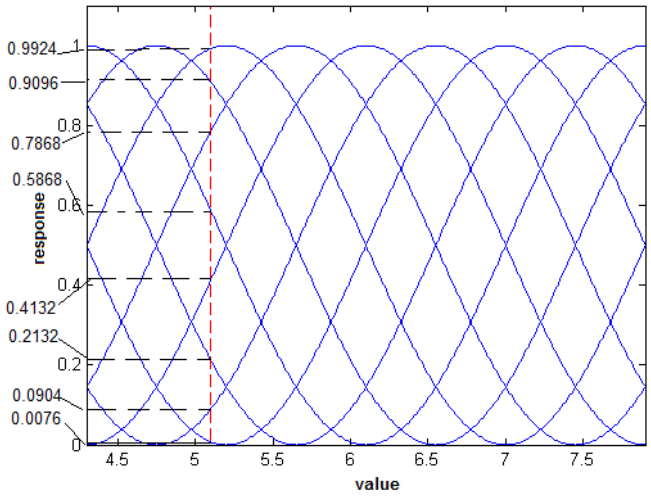

Fig. 2. Encoding of a real valued feature of 5.1 using 8 Square Cosine neurons.

It can be seen that there are 8 spiking times resulted from both Gaussian and square cosine population encoding schemes for the value of 5.1; however, the Square Cosine population encoding scheme results in earlier spike times. Each spiking time is represented by a neuron in the encoding layer. In the following experiments, the effect of these two population encoding methods on SpikeTemp performance and efficiency will be evaluated.

Time-to-first spike decoding is employed at the output layer where an input sample is considered to be correctly classified if the first spike is produced at an output neuron whose class label matches the class label of the current input sample; otherwise an input sample is considered to be incorrectly classified.

C. Network Topology

Fig. 3 presents the network topology that consists of a layer of encoding neurons and a layer of output neurons. The neurons in the encoding layer convert the input features to a set of spiking times using Gaussian Receptive Field / Square Cosine population encoding. The parameter q represents the number of Gaussian receptive fields / Square Cosines and its value is chosen by trial and error. Each neuron in this layer provides a spike time which is fed to the next layer, and is fully connected to the neurons in the output layer. The number of neurons in this layer is determined by the dimensionality of the dataset and the value of the parameter q. For instance, if the dimensionality of the dataset is denoted by m then the size of the encoding layer is given by $m*q$. The set of spiking times represented by these neurons in the encoding layer is in the range of the time coding interval $[0,T_{ref}]$ and T_{ref} = 9 ms.

The output layer has no output neurons at the beginning of the training process. A new output neuron is added dynamically when an incoming sample is received and is fully connected to the neurons in the encoding layer. Every added

output neuron is assigned a class label corresponding to the label of the incoming sample during training. Only one synaptic connection exists between every encoding neuron and every output neuron. Multiple sub-connections are not used in this work.

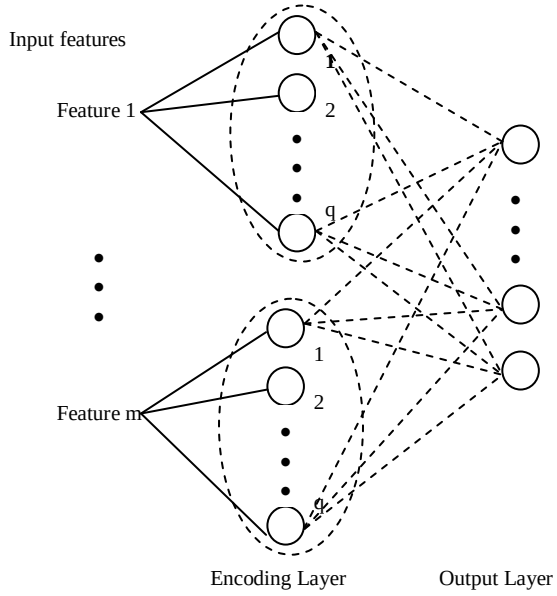

Fig. 3. A feedforward SNN with adaptive output layer.

III. Learning and Network Structure Adaptation

Delorme and Thorpe [30] proposed a learning method to update weights based on the rank order of the incoming spikes that was trained in an offline mode as shown in (3):

(3)

Where Δw_{ji} is the change of weight between neuron j in the encoding layer and neuron i in the output layer.
mod is a constant, representing the modulation factor that is in the range of [0 1].
order(X_j) is the relative order based on the spiking times for afferent neuron j. For example, the first spike arrives (order(X_j) = 0), the second spike arrives (order (X_j)= 1), the third spike arrives (order (X_j)= 2) and so on.
Tr is the number of samples for the training dataset.

Wysoski et al. [33] adapted the offline learning procedure in [30] to online learning with an adaptive network structure. The learning method used to train the weights in [33] is based on the rank order of the incoming spiking arrival and hence the precise timing information is thrown away. Weight change Δw is only determined by the mod term as shown in (4):

(4)

In SpikeTemp, the precise spike time is exploited to update the weights. Equation 5 describes the change in the weight of the synapse connecting an encoding neuron j to an output neuron i, where t_j denotes the precise spiking time of the encoding neuron j instead of the order of spiking times as in [33], such that earlier firing times invoke a larger weight change, τ represents a time constant. This weight change is added to the initial baseline weight.

(5)

A. Comparison between SpikeTemp and the rank-order based learning approaches

To illustrate the difference between SpikeTemp and the rank-order based approaches consider a simple SNN that consists of a layer of three encoding neurons and a layer of one output neuron which receives two input vectors (2.0, 5.0, 8.0) and (1.1, 2.0, 8.2) respectively. The three encoding neurons are fully connected to the output neuron.

Using a rank order approach the rank ordering stays the same and the learning algorithm maintains a constant weight update for both input patterns. Therefore, the maximum postsynaptic potential for this output neuron, which is computed using (1) when all three input spikes have been used, is the same for both input patterns. However, with the SpikeTemp approach the different spike times inherently cause different weight changes, and different maximum postsynaptic potential for this output neuron for the two input patterns. As a result, the weight changes and the maximum postsynaptic potential now correlate better with the input pattern, which contributes to an improved learning performance. Furthermore as SpikeTemp approach removes the need to rank all the spikes in a window, it reduces the computational effort with respect to the rank order approach.

B. Learning Procedure

The overall aim of the proposed supervised procedure is to map a set of input samples to a set of classes. The weights are updated after each sample is propagated into the network. The following sequential steps describe the learning procedure of SpikeTemp:

1) Each real-valued feature of a data sample is encoded using q Gaussian receptive fields/Square Cosine encoders. An output neuron is then created and all weights between every neuron in the encoding layer and this added output neuron are initialised to a constant 0.1 (this value was chosen by systematic experiments based on classification performance, please see section VI.A). The weights are updated when an incoming sample is propagated into the network using (5).

2) The maximum postsynaptic potential for this added output neuron, $(PSP_{max}(i))$, is calculated using (1) when all input spikes have occurred. The firing threshold of this added output neuron $(PSP_{th}(i))$ is a fraction of the maximum postsynaptic potential represented by (6), so similar samples can trigger an output spike.

(6)

con is a constant ($0< con < 1$) that represents how similar a sample can trigger an output spike. During training, the class label of this new added output neuron is set as the label of the current sample, this is a supervised process.

3) The similarity between this new added output neuron *i* and other output neurons that have the same class label is calculated. It is defined as the inverse of the Euclidean distance between the weights of this newly added output neuron and other output neurons (*k*) that belong to the same class. If the similarity with one of the existing output neurons is greater than a predefined threshold, then the newly added output neuron is merged with the output neuron whose synaptic weights are most similar to it. The SNN structure is then updated. Currently this threshold is set based on performance, in terms of accuracies by trial and error in the range of [0.5 1.0], please see detailed discussion in section VI.C. For example if the i^{th} output neuron was merged with the k^{th} output neuron, the resultant weights w_{jk} and firing threshold of this merged neuron k in the output layer are determined by (7) and (8) respectively. The total trained number of samples for output neuron *k*, N_k , is then incremented by one. The above mentioned rules for pruning output neurons were inspired by [33].

(7)

(8)

4) In order to evaluate the performance of the system, after the one pass training, the weights were fixed and the training dataset and testing dataset were fed into the network so that the classification accuracy on both training and testing datasets can be calculated. There are no weight updates and neuron merging during testing as weight updates and neuron merging happen during training only. The spikes from all neurons in the encoding layer are integrated by each neuron in the output layer and when the threshold of an output neuron has been reached, a spike is emitted. When an output neuron fires first, if the class label represented by this output neuron matches the class label of this sample, we treat it as correctly classified. Otherwise, we treat it as incorrectly classified. During the testing stage, once a spike is emitted in the output layer, the simulation for this sample is finished, and another sample can be presented to the network. The supervision mechanism is used to set the class label of every new added output neuron as the label of the incoming sample, and similarity is calculated between the output neurons which have the same class label during training so that pruning can only occur on the output neurons that have the same class label.

IV. Benchmarking: Experiments and Results

In this section selected benchmark datasets, from the UCI Machine Learning Repository, were used to test and evaluate the performance of SpikeTemp, employing either Gaussian or Square Cosine population encoding. The results obtained from SpikeTemp have also been compared with the results obtained using the rank order based learning rule used in [33][34] and standard classical machine learning methods. Lateral inhibition in the simulations of rank-order and SpikeTemp learning is not used in the following experiments.

A. Data description and simulation results for SpikeTemp and rank order method

The following benchmark datasets from the UCI Machine Learning Repository are used: Pima diabetes, Bupa Liver disorders, Ionosphere, Wisconsin Breast Cancer (WBC), Image segmentation, Abalone, Yeast, EEG eye state and IRIS. Each dataset is divided into training (Tr) and test (Ts) sets as outlined in Table I.

TABLE I
DESCRIPTION OF DIFFERENT DATASETS

Database	T	Tr	Ts	m	c	q	N
IRIS	150	90	60	4	3	30	120
WBC	683	455	228	9	2	15	135
Image	2310	210	2100	18	7	10	180
Abalone	4177	2000	2177	7	3	7	49
Pima Diabetes	768	512	256	8	2	10	80
Liver Disorder	345	230	115	6	2	25	150
Ionosphere	351	234	117	33	2	7	231
Yeast	1484	990	494	8	10	10	80
EEG eyeState	14980	9990	4990	14	2	10	140

Table I summarises the properties of each dataset: size of dataset (T), number of training (Tr) and testing (Ts) samples, number of features or the dataset dimension (m), the number of classes presented in each dataset (c), the number of Gaussian receptive fields /Square Cosine curves for each dataset (q) and the size of the encoding layer (N). For example, the Wisconsin breast cancer dataset consists of 699 instances with 9 feature values: 16 samples of the 699 instances that have missing data are removed in this experiment for simplicity. So there are 683 samples remaining that are divided into two sets (the first 455 samples for training and the remaining 228 samples for testing). Each feature value is encoded with 15 Gaussian receptive fields /Square Cosine curves resulting in a total of 135 neurons in the encoding layer (encoding layer, 15*9 neurons). In the Image Segmentation dataset, there are 19 continuous feature values, but since the fourth feature values are similar for all the samples, this feature is removed for simplicity. Each feature value of the remaining 18 feature values is encoded with 10 Gaussian receptive fields /Square Cosine curves resulting in a total of 180 neurons in the encoded layer (encoded layer, 10*18 neurons). In the Ionosphere dataset, there are 34 numeric feature values, since the second feature values are also similar for all the samples, this feature is also removed for simplicity. Each feature value of the remaining 33 feature values is encoded with 7 Gaussian receptive fields /Square Cosine curves resulting in a total of 231 neurons in the encoded layer (encoded layer, 7*33 neurons). Please see section VI.D for detail of how to decide the number of the Gaussian receptive fields for each dataset.

TABLE II
CLASSIFICATION PERFORMANCE IN TERMS OF SIMULATION TIMES AND SIZE OF OUTPUT LAYER FOR SPIKETEMP AND RANK ORDER METHOD USING GAUSSIAN/SQUARE COSINE POPULATION ENCODING

Database	SpikeTemp				Rank order method			
	Gaussian		Square Cosine		Gaussian		Square Cosine	
	T_{sim}(min)	Num_o	T_{sim}(min)	Num_o	T_{sim}(min)	Num_o	T_{sim}(min)	Num_o
IRIS	0.85	87	0.15	63	11.7	84	1.58	58
WBC	3.59	306	1.13	284	209.8	280	58.2	265
Image	6.82	174	1.69	130	367.1	191	41.4	114
Abalone	6.56	39	2.03	46	66.0	16	14.6	13
Pima Diabetes	5.21	431	0.76	365	191.6	265	37.0	187
Liver Disorder	1.65	226	0.31	170	149.4	215	9.67	157
Ionosphere	3.11	223	0.26	210	36.2	213	12.2	215
Yeast	14.3	549	0.80	300	307.9	184	74.3	258
EEG eyeState	6.56	7	5.42	5	60.7	5	21.0	5

TABLE III
COMPARISON OF THE CLASSIFICATION ACCURACY OF SPIKETEMP WITH A SELECTION OF MACHINE LEARNING APPROACHES

Database	SpikeTemp		Rank order approach		KNN	MLP	SVM
	(Gaussian)	(Square Cosine)	(Gaussian)	(Square Cosine)	(k=3)		
			Acc_TR(%)/ Acc_TS(%)				
IRIS	**100/96.7**	100/95.0	100/95.0	98.9/93.3	96.0/92.0	100/94.8	100/96.7
WBC	99.1/98.3	99.6/92.1	**99.6/98.7**	92.5/89.9	96.9/98.7	97.4/90.9	96.7/98.2
Image	89.1/82.0	91.9/84.4	71.9/70.9	86.7/80.0	96.7/86.3	71.6/52.1	**91.4/87.6**
Abalone	45.7/47.8	52.2/52.0	44.5/44.8	53.4/51.7	82.2/59.3	**67.4/60.5**	50.4/51.7
Pima diabetes	77.5/67.6	91.2/70.3	81.0/61.7	79.9/67.6	84.4/69.9	84.5/76.2	**79.3/80.5**
Liver disorder	93.0/58.3	80.4/52.2	86.5/59.1	78.7/56.5	**81.7/67.8**	89.8/59.0	100/65.2
Ionosphere	86.8/91.5	**92.7/95.7**	81.6/74.4	85.9/70.9	90.2/95.7	99.5/83.6	100/85.5
Yeast	56.7/31.6	53.5/37.2	50.5/31.4	53.3/31.2	**84.0/51.6**	53.2/35.9	40.8/33.0
EEG eyeState	**55.4/54.6**	55.5/54.4	55.4/54.6	55.5/54.4	99.9/53.9	86.0/53.5	88.8/53.0

In the following tables, Acc_Tr/Acc_Ts represents the training/testing classification accuracy; Num_o represents the total number of neurons in the output layer after training; T_{sim} is the running time in minutes which represents the complete simulation time including calculation of training and test accuracies. The simulations are run on a laptop with the following specifications: *Intel Core 2 Duo, 2.17GHz, 2 GB RAM.*

Table II shows the classification performance of SpikeTemp and rank order method for different datasets in terms of running time and the number of the output neurons after training using Gaussian receptive fields /Square Cosine population encoding. Classical methods such as the k-nearest neighbour algorithm (KNN), Support Vector Machine (SVM) and Multi-layer Perceptron Neural Network (MLP), are popular machine learning techniques and are used to benchmark the performance of SpikeTemp. Table III compares the classification performance of these selected datasets using SpikeTemp with rank order method, KNN (K is set to 3), SVM (Quadratic kernel function) and MLP. The results listed in Table III using the MLP network are based on 15 neurons in the hidden layer and 1 neuron in the output layer, and the network is trained using Levenberg-Marquardt backpropagation. The number of hidden neurons is chosen by trial and error in terms of accuracy performance. A comparable level of performance has been reached for these datasets across all methods. The reported results using Gaussian receptive fields population encoding are obtained by employing *con=0.45*, *Th_sim=0.8*, *τ=40* for SpikeTemp and by employing *con=0.45*, *Th_sim=0.8*, *mod=0.98* for rank order method. These values were selected following analysis of trial experiments, please see section VI. The reported results using Square Cosine population encoding are obtained by employing *con=0.45*, *Th_sim=0.8*, *τ=3* for SpikeTemp, and by employing *con=0.45*, *Th_sim=0.8*, *mod=0.98* for rank order method.

B. Analysis of results

From Table II and Table III we can see that both the rank-order and SpikeTemp methods have the ability to classify a wide range of datasets with various dimensions and number of classes after just one presentation of the training samples. However, the results in Table III show that the SpikeTemp classification accuracy is better on datasets such as Image, Abalone, Pima Diabetes and Ionosphere as compared to the rank order method using Gaussian receptive fields population encoding and a comparable performance is achieved on the remaining datasets. The results in Table III show that a comparable performance is achieved on the datasets when square cosine population encoding is employed. Furthermore, the results in Table II show that the simulation time of the SpikeTemp network is much faster than that of the rank order based approach on the selected datasets as the computational overhead is much lower.

From Table II and Table III, we can see that for SpikeTemp, better accuracy is achieved for the Image, Abalone, Pima Diabetes and Ionosphere datasets when Square Cosine population encoding is employed, and less output neurons are added after training using square cosine encoders as compared to Gaussian encoders except the Abalone dataset. The simulation time of the network is much faster when Square Cosine encoders are employed for these selected datasets. A detailed comparison of the resulting eight spiking times

produced by both approaches (please see section II.B) illustrates that the Square Cosine encoder results in earlier spike times. Note an early spike time results in a faster simulation because the simulation is terminated when a spike is produced in the output layer.

Although both the time coding and rank order coding are dependent on timing of spikes the former has a much greater information capacity [39]. The use of the absolute time of a spike over a set of N neurons can provide $\log_2 N^N$ bits of maximum amount of information capacity [39]. If one neuron can have only one spike, N^N possible timing of spikes are used to encode patterns. In contrast a rank order method only uses the relative time of spikes over a set of N neurons, and thus can provide $\log_2 N!$ bits of maximum amount of information capacity if one neuron can have only one spike, N! possible order of spiking are used to encode patterns. Thus time coding can represent a greater amount of information capacity than the rank order coding.

In sub-sections IV.C and IV.D, we use the popular IRIS and WBC datasets, to depict weight distribution after training and dynamic changes of the sizes of the output layer during training for SpikeTemp and the rank order methods using Gaussian receptive fields population encoding.

C. *Weight distribution after training*

For the IRIS dataset, the distribution of the final updated weights for every added output neuron after training for SpikeTemp and the rank order approach are depicted in Fig. 4 (a) and (b), respectively. For the WBC dataset, the distribution of the final updated weights for every added output neuron after training for SpikeTemp and the rank order approach are depicted in Fig. 5 (a) and (b), respectively. From these figures, we can see that the proposed learning rule and the rank order learning method have quite different effect on the updated weights distribution after training. This is due to the difference between weights adjustments in both methods. In rank order method, it is the order of spike timings that is

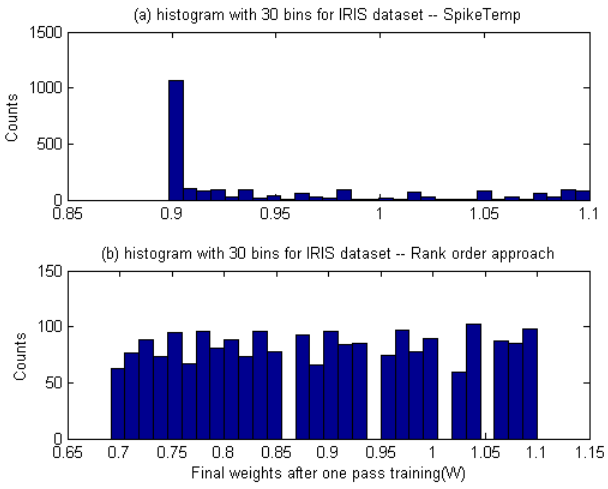

Fig. 4. Weight distribution after training with the proposed learning method (a) and the rank order method (b) for the IRIS dataset.

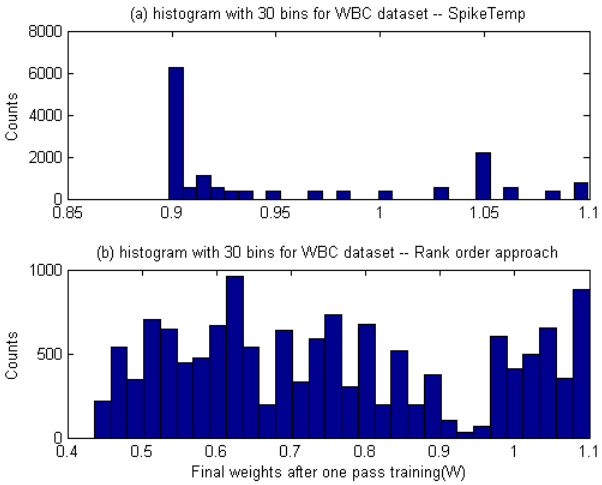

Fig. 5. Weight distribution after training with the proposed learning rule (a) and rank order method (b) for the WBC dataset.

important for the weights update while in SpikeTemp it is rather the precise timing of spikes that contributes to the weights update.

D. *Evolution of the size of the output layer*

The dynamic evolution of the number of output neurons during training on the IRIS and WBC datasets is illustrated in Fig. 6 (a) and (b), respectively. It can be seen how output neurons are dynamically changed as more training samples are presented to the network. As described in section II.C and III.B, a new output neuron is added dynamically when an incoming sample is received, then if the similarity between this new added output neuron with one of the existing output neurons is greater than a predefined threshold, the newly added output neuron is merged, so if the number of output neurons does not increase linearly as a new sample is propagated into the network, then it implies that an output neuron has been pruned. This pruning effect is more pronounced in Fig. 6 (b) as compared to Fig. 6 (a).

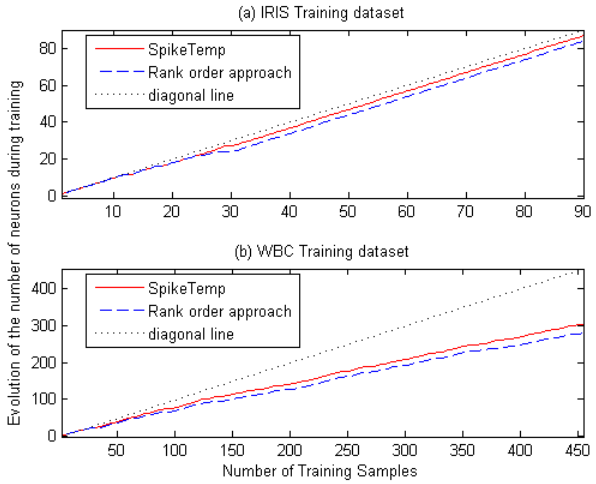

Fig. 6. Changes of number of output neurons against the number of training samples during training for IRIS (a) and WBC (b) datasets (*con=0.45, Th_sim=0.8, τ=40, mod=0.98*). The black, dotted, diagonal line represents a linear addition of output neurons, which makes it easier to see the pruning effect.

V. Application of SpikeTemp to face recognition

This section describes the application of SpikeTemp to visual pattern recognition using the AT&T dataset. This dataset was used to test and evaluate the performance of the rank-order method in [33] using PCA to extract the features and is available from [40]. The results are again compared with both the rank-order approach and classical machine methods.

A. Data preparation and simulation results

The AT&T dataset consists of 400 greyscale face images (92 x 112 pixels) corresponding to 40 persons such that each person has 10 face views. Images for some individuals were taken at different sessions, so light conditions and facial expressions are not systematically controlled. Figure 7 depicts three face image samples, one frontal view, one view taken from the left side (30°) and one view taken from the right side (-30°). The topology of SpikeTemp is only suitable for a 1D inputs so the features of these face images have to be presented in this format. The approach employs a linear transformation of the input image to extract its principal components using PCA (principal component analysis) function in Matlab which are then presented to the network. An increase in the number of principal components results in a better classification performance until it reaches a fixed value. In the following experiments, the number of principal components selected was 20 and seven samples from each individual were used for training; the remaining three samples of each person were used for testing, so a set of 280 image samples is used for training and a set of 120 image samples is used for testing. Each feature value is encoded with 10 Gaussian receptive fields /Square Cosine curves resulting in a total of 200 neurons in the encoded layer (encoded layer, 20*10 neurons).

Fig. 7. Example of three face image samples (frontal, 30° and -30°) [40].

Tables IV and V list the number of output neurons after training, the classification performance for the AT&T dataset using SpikeTemp and Rank order method and the running times that include calculation of training and testing accuracies when the similarity threshold value is taken as 0.2, 0.25, 0.3, 0.5 and 0.8 respectively, while the values of the parameters con and τ are kept the same. The results in Table IV are derived from a network that used the Gaussian receptive fields based encoding scheme whereas the results in Table V the square cosine encoding scheme is used instead.

Increasing the similarity threshold (*Th_sim*) increases the total number of output neurons required which has an impact on the classification performance. This is because the similarity threshold (*Th_sim*) determines if an added output neuron should be merged or not, so a smaller similarity threshold increases the likelihood of an added output neuron being merged. After comparing the results in Table IV and Table V, we can see for the AT&T dataset that not only the performance of SpikeTemp is better than that of rank order method [33], but also the simulation time of the network using SpikeTemp is much faster than using the rank order method.

TABLE IV
Results with different similarity threshold (*Th_sim*) for SpikeTemp and Rank order approach (con=0.45; 10 Gaussian receptive fields)

	Th_sim	0.2	0.25	0.3	0.5	0.8
PCA+SpikeTemp (τ=25, Gaussian)	Num_o	53	100	152	266	280
	Acc_Tr	94.6	99.6	100	100	100
	Acc_Ts	**86.7**	**89.2**	**92.5**	**93.3**	**93.3**
	T_{sim}(min)	0.920	1.37	1.75	3.05	3.56
PCA+Rank order method (mod=0.98,Gaussian)	Num_o	40	45	77	185	257
	Acc_Tr	95.4	97.9	98.6	100	100
	Acc_Ts	80.8	85.0	82.5	88.3	88.3
	T_{sim}(min)	13.8	14.9	22.0	30.5	45.8

TABLE V
Results with different similarity threshold (Th_sim) for SpikeTemp and Rank order approach (con=0.45; 10 Square cosine curves)

	Th_sim	0.2	0.25	0.3	0.5	0.8
PCA+SpikeTemp (τ=3, Square Cosine)	Num_o	50	93	132	230	266
	Acc_Tr	96.8	99.3	99.6	100	100
	Acc_Ts	**84.2**	**86.7**	**89.2**	**90.8**	**90.8**
	T_{sim}(min)	0.465	0.596	0.602	0.707	1.59
PCA+Rank order Method(mod=0.98, Square Cosine)	Num_o	40	55	99	222	269
	Acc_Tr	95.7	97.1	99.6	100	100
	Acc_Ts	78.3	83.3	**89.2**	87.5	87.5
	T_{sim}(min)	2.49	2.75	3.05	5.01	5.69

B. Comparison between SpikeTemp and other machine learning methods

To further assess the performance of SpikeTemp, it is compared with other existing classifiers used in face recognition, namely SVM, MLP and SNN. Table VI summarises the obtained comparison results.

TABLE VI
Compares among different methods of face recognition (con=0.45, T=25, mod=0.98, 10 Gaussian receptive fields)

Methods	Acc_Ts (%)	Num_o/Th_sim	Property
PCA+SpikeTemp	93.3	266/0.5	One-pass online
	92.5	152/0.3	One-pass online
	89.2	100/0.25	One-pass online
PCA+Rank order method	88.3	185/0.5	One-pass online
PCA+SVM [33]	90.7		Batch mode
PCA+MLP [33]	89.6		Batch mode
PCA+KNN (k=3)	92.5		Batch mode

C. Analysis of results

The results in Table VI clearly show that after just one presentation of the training samples, SNNs trained with the proposed learning approach SpikeTemp outperforms the rank order method for this face recognition dataset, with comparable performance to other machine learning offline methods.

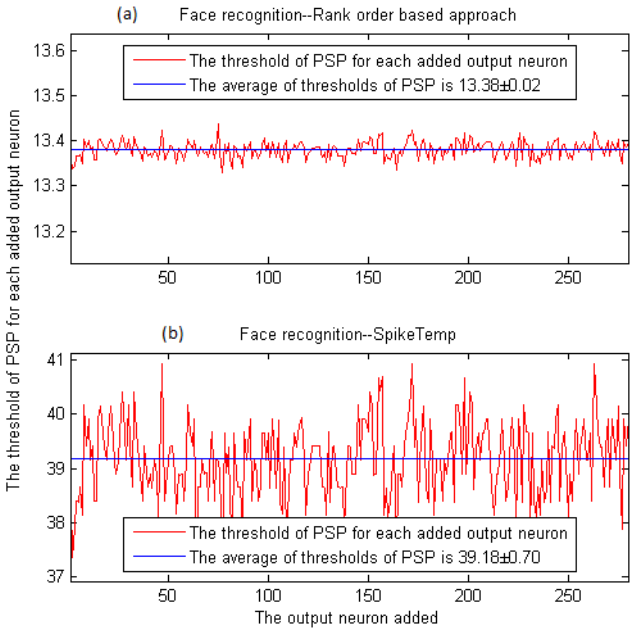

Fig. 8. The PSP threshold (P) for each added output neuron for (a) the rank order approach (b) SpikeTemp.

As described in the previous section, the parameter similarity threshold (*Th_sim*) determines how many output neurons are added after training. In the following experiments, *Th_sim* is set to 0.8 or more for SpikeTemp, and is set to 1.0 or more for rank order method so that it is enough to add an output neuron for every sample for both SpikeTemp and the rank order method. The dynamic characteristics of the PSP thresholds (PSP_{th}) used for each added neuron during training are shown in Fig. 8 (a) and (b) for the rank order approach and the proposed SpikeTemp, respectively.

The average of these PSP thresholds for the proposed SpikeTemp is 39.18±0.70, and the average of these PSP thresholds for the rank order method is 13.38±0.02 (see blue straight lines in Fig. 8). The difference of standard deviations between SpikeTemp and the rank order method clearly shows that the thresholds of PSP for each added output neuron for SpikeTemp oscillate a lot more around the average value than the rank order method; this might be the reason that SpikeTemp outperforms the rank order method for face recognition dataset.

VI. Evaluation of Various Parameters Effect on the Learning Performance

In this section we use two popular datasets, namely the IRIS and WBC, to explore the effect of different parameters on the performance of SpikeTemp and the rank order methods using Gaussian receptive fields population encoding. These parameters include the initial weight value, the con value, the threshold value (*Th_sim*) for merging neurons, the τ value and the number of Gaussian receptive fields/Square cosine chosen (q value). In addition, the robustness of these methods to white noise is evaluated and compared.

A. *Effect of initial weight values*

Experiments were carried out to explore the effect of initial weight values w_{ij} between the encoding layer and the output layer on the performance of SpikeTemp and the rank order approach. Fig. 9 (a) and (b) shows the change in classification performance with respect to the initial weight values between the encoding layer and the output layer for IRIS and WBC datasets, respectively. In the experiments reported earlier, the initial value of the weights between the encoding layer and the output layer is set to 0.1. The results shown in Fig. 9 (a) indicate that the obtained accuracy performance for the IRIS dataset degrades as the initial weight w_{ij} is increased, while the classification accuracies remain above 95% for weight values w_{ij}<=1.0 for SpikeTemp, and weight values w_{ij}<=0.7 for the rank order approach. The results shown in Fig. 9 (b) indicate that the obtained accuracy performance for the WBC dataset degrades as the initial weight w_{ij} is increased, while the classification accuracies remain above 95% for weight values w_{ij}<=0.8 for SpikeTemp, and weight values w_{ij}<=1.8 for the rank order approach.

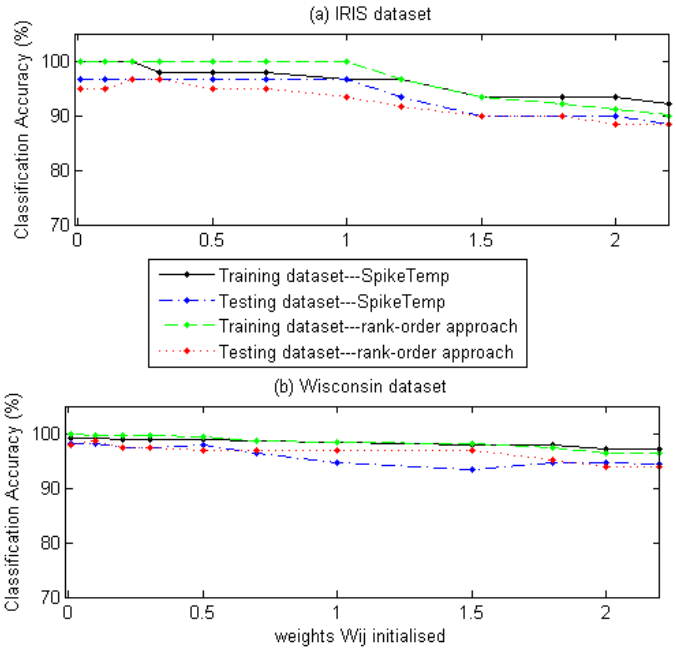

Fig. 9. Classification accuracies for (a) IRIS and (b) WBC datasets against the initial values of the weights between the encoding layer and the output layer.

B. *Effect of the maximum PSP fraction (con)*

Fig. 10 (a) and (b) show the changes in classification performance with respect to the *con* values for IRIS and WBC datasets respectively. For both SpikeTemp and the rank order method, the results using Gaussian population encoding shown in Fig. 10 indicates that the classification accuracies for both training and testing datasets remain above 95% when the value of the parameter con is set in the range of [0.45 0.55] for the IRIS dataset, and is set in the range of [0.35 0.60] for the WBC dataset. The classification accuracies gradually degrades when the value of the parameter *con* is greater than

0.55 in the case of IRIS and when it is greater than 0.60 in the case WBC.

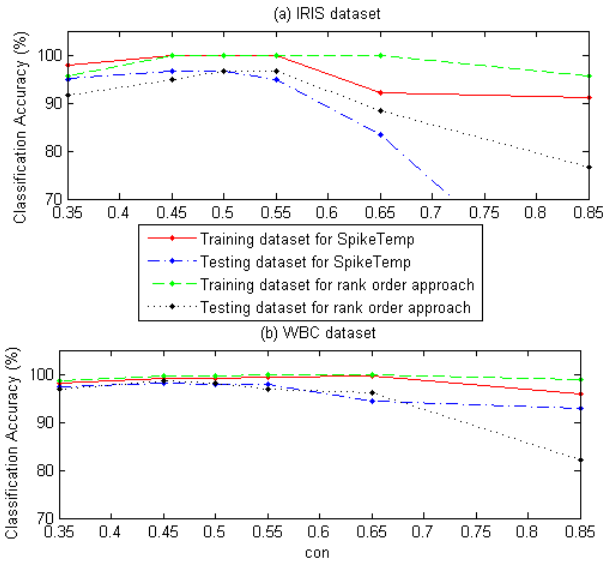

Fig. 10. Classification accuracies for IRIS (a) and WBC (b) datasets against the *con* value.

C. *Effect of the pruning threshold (Th_sim)*

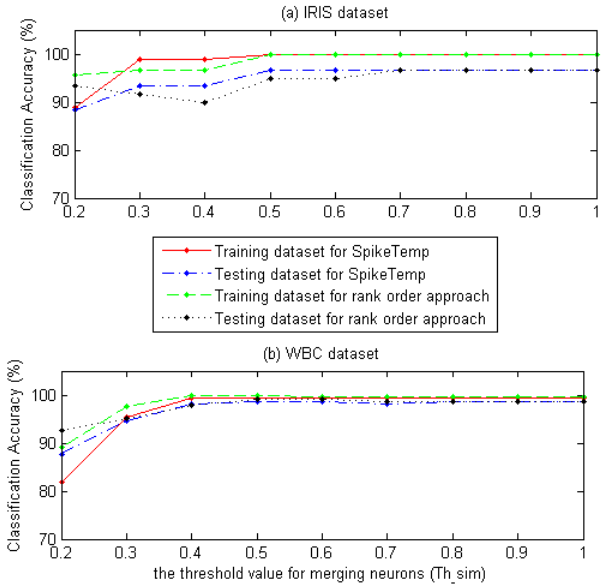

Fig. 11. Classification accuracies for IRIS (a) and WBC (b) datasets against the threshold value for merging neurons.

The Changes in classification performance with respect to the threshold value (*Th_sim*) for merging neurons are shown in Fig. 11 (a) and (b) for IRIS and WBC datasets, respectively. For both SpikeTemp and the rank order methods, the results using Gaussian population encoding shown in Fig. 11 indicate that the accuracy for both training and testing datasets is maintained above 95% when *Th_sim* is greater than 0.5 for the IRIS dataset, and when it is greater than 0.4 for the WBC dataset. A smaller threshold value increases the likelihood of an added output neuron that should not be merged being merged, the classification accuracies degrades gradually when the threshold value is less than 0.5 in the case of the IRIS dataset and when it is less than 0.4 in the case of the WBC dataset.

D. *Effect of the number of receptive Gaussian fields/Square cosines (q)*

The changes in classification performance with respect to the number of Gaussian receptive fields (q value) are shown in Fig. 12 (a) and (b) for IRIS and WBC datasets, respectively. The results using Gaussian population encoding indicate that the best performance is obtained when the number of Gaussian receptive fields is set to 30 for the IRIS dataset, and is set to 15 for the WBC dataset for both SpikeTemp and the rank order methods. From Fig. 12 (a) we can see that for IRIS dataset the accuracies for training datasets are maintained above 95% when q is in the range of [20, 50] for both SpikeTemp and the rank order method. For WBC dataset, Fig. 12 (b) shows that training and testing accuracies for both SpikeTemp and the rank order methods are maintained above 95% when q is in the range of [10, 25]. However, the performance degrades when q is less than 10.

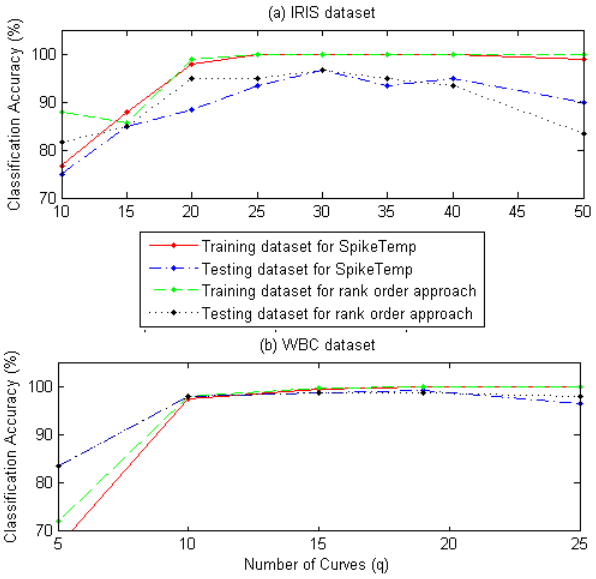

Fig. 12. Classification accuracies for IRIS (a) and WBC (b) datasets against the number of Gaussian receptive fields.

E. *Effect of time constant (τ)*

The changes in classification performance for the IRIS and WBC datasets with respect to the value of τ for SpikeTemp and the value of *mod* for the rank order method are shown in Fig. 13 (a) and (b), respectively. Fig. 13 (a) shows that the classification accuracies remain above 95% when τ is set in the range of [20 50]. Fig. 13 (b) shows that the value of *mod* has little effect on the classification accuracies for rank order method using Gaussian population encoding.

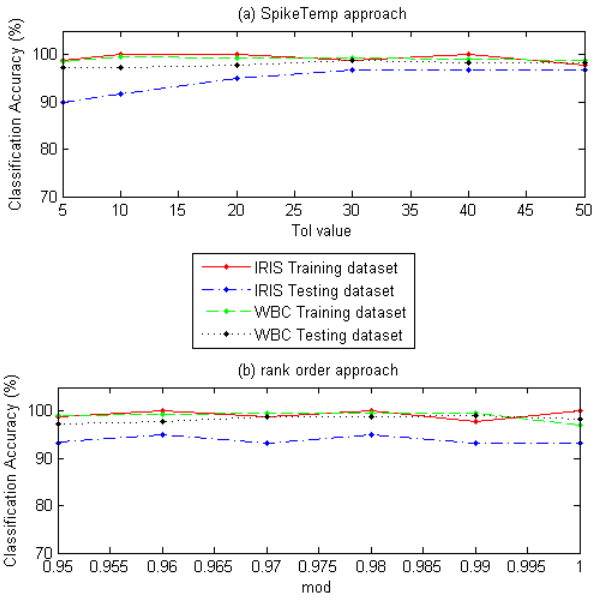

Fig. 13. Classification accuracies against τ (a) and (b) values using Gaussian population encoding.

F. Robustness to input noise

Noise analysis is conducted by adding different levels of additive white Gaussian noise to the IRIS, WBC and Pima Diabetes datasets. It is worth noting that the 'noiseless' data described in section 4 was used to train the adaptive SNN, and the SNN produced was employed to test the classification accuracies of the noisy training dataset and testing dataset. We conducted ten experimental trials for each level of additive white Gaussian noise where the average performance is computed.

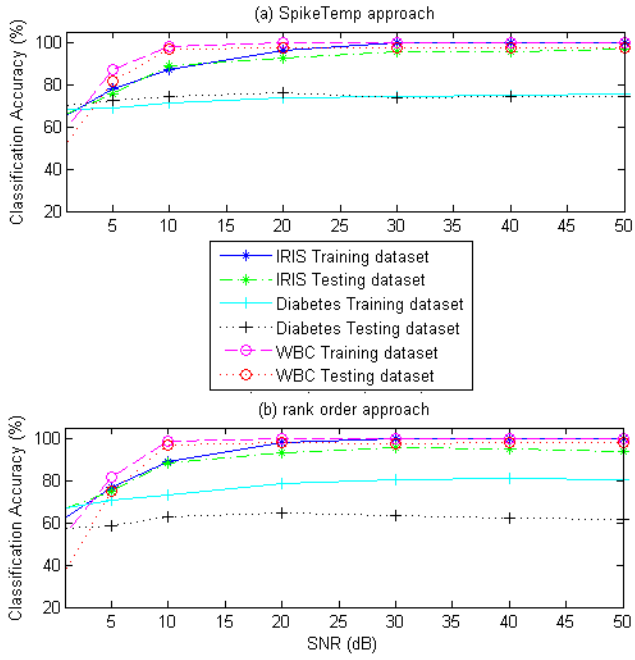

Fig. 14. SpikeTemp (a) and rank order method (b) classification accuracies with respect to different signal to noise ratios (SNR).

Fig. 14 (a) and (b) show the averaged training and testing accuracies of ten runs obtained in each level of additive white Gaussian noise when different levels of noise were added to the IRIS, WBC and Pima Diabetes datasets for SpikeTemp and the rank order method, respectively. For the IRIS dataset, it can be seen that the classification accuracies for both training and testing sets remain above 90% for signal to noise ratio (SNR) up to 12 dB for both SpikeTemp and rank order methods. For the WBC dataset, it can be seen that the classification accuracies for both training and testing sets remain above 96% for signal to noise ratio (SNR) up to 10 dB for both SpikeTemp and the rank order method. As for the Pima Diabetes dataset, it can be seen that the classification accuracies for training datasets remain above 70% for signal to noise ratio (SNR) up to 10 dB; the classification accuracies for testing datasets remain above 70% for signal to noise ratio (SNR) up to 1dB for SpikeTemp. However, for the rank order method, the classification accuracies for training datasets remain above 70% for signal to noise ratio (SNR) up to 5 dB and the classification accuracies for testing datasets remain above 60% for signal to noise ratio (SNR) up to 10 dB. These results indicate noise robustness for both the proposed method and the existing rank order approach.

VII. Conclusion

This paper presents an enhanced rank-order based learning approach (SpikeTemp) for SNNs with an adaptable structure where the learning method is based on the precise times of incoming spikes which removes the need to explicitly rank the order of the incoming spikes. As a result, SpikeTemp is a more efficient than the rank order learning method. The neurons in the encoding layer temporally encode real valued feature vectors into spatio-temporal spike patterns, and output neurons, which process spatio-temporal inputs from the encoding layer, are dynamically grown and pruned as new spatio-temporal spiking patterns are presented to the spiking neural network. The proposed SpikeTemp approach was benchmarked on a selection of datasets from the UCI machine learning repository and on an image recognition task. It was shown that SpikeTemp can classify different datasets with improved accuracy and simulation times than those of the rank order method. As for the rank-order learning method, SpikeTemp is scalable for a wide range of datasets with various dimensions and numbers of classes and is more efficient than the existing rank order learning method.

Both the Gaussian receptive fields and the Square Cosine population encoding methods are employed to convert input data into a set of spiking times. The results show that the simulation time of the network using the Square Cosine population encoding methods is shorter and the number of output neurons added for most of the datasets is much smaller. The results are also compared with existing machine learning algorithms and it was shown that SpikeTemp is able to classify different datasets after just one presentation of the training samples with comparable classification performance. In addition, SpikeTemp allows the detection of new classes without forgetting those that were previously learned. Furthermore, as each sample is handled only once and there is

no need to repeatedly re-feed the training set unlike the classical classifiers, this makes SpikeTemp computationally efficient and thus more widely applicable.

The trained feed-forward SNN in SpikeTemp consists of two layers of spiking neurons, a population of output neurons are added to encode each class, a large number of output neurons are added/required in the output layer to represent each class after training. Future work will explore the alternative approaches to reduce the neuron count and yet expand the network topology of SpikeTemp.

REFERENCES

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: MacMillan Publishing Company Press, 1994.

[2] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge University Press, 2002.

[3] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer, IEEE*, vol. 29, no. 3, pp. 31- 44, Mar. 1996.

[4] R. Van Rullen, R. Guyonneau, and S. J. Thorpe, "Spike times make sense," *TRENDS in Neurosciences*, vol. 28, no. 1, pp. 1-4, 2005.

[5] W. Maass, "Fast sigmoidal networks via spiking neurons," *Neural Computation*, vol. 9, pp. 279-304, 1997.

[6] S. M. Bohte, "The Evidence for Neural Information processing with Precise Spike-times: A survey," *Natural Computing*, vol. 3, no. 2, pp. 195-206, 2004.

[7] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," in *Advances in Neural Information Processing Systems*, M. Mozer, M.I. Jordan, and T. Petsche, Eds, Cambridge, MA: The MIT press, vol. 9, pp. 211-217, 1996.

[8] S. J. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, no. 6-7, pp. 715-726, 2001.

[9] S. M. Bohte, J. N. Kok, and H. L. Poutre, "Spike-prop: error backprogation in multi-layer networks of spiking neurons," in *Proceedings of the 8th European Symposium on Artificial Neural Networks ESANN'2000*, Bruges, Belgium, Apr. 2000, pp. 419-425.

[10] S. M. Bohte, J. N. Kok, and H. L. Poutre, "Error-backprogation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17-37, 2002.

[11] S. Moore, *Back-Propagation in Spiking Neural Networks*. M.S. thesis, University of Bath, U.K., 2002.

[12] B. Schrauwen and J. V. Campenhout, "Improving SpikeProp: Enhancements to an Error-Backpropagation Rule for Spiking Neural Networks," in *Proceedings of the 15th ProRISC Workshop*, Veldhoven, Netherlands, Nov. 2004, pp. 301-305.

[13] S. McKennoch, D. Liu, and L. G. Bushnell, "Fast modifications of the SpikeProp algorithm," in *Proceedings of the 2006 IEEE International Joint Conference on Neural Networks*, Vancouver, BC, Canada, Jul. 2007, pp. 3970-3977.

[14] Q.X.Wu, T.M. McGinnity, L.P. Maguire, B. Glackin, and A. Belatreche, "Learning under weight constraints in networks of temporal encoding spiking neurons," *Neurocomputing*, vol. 69, no. 16-18, pp. 1912-1922, 2006.

[15] S. M. Silva and A. E. Ruano, "Application of the Levenberg-Marquardt method to the training of spiking neural networks," in *Proceedings of the 2006 IEEE International Joint Conference on Neural Networks*, Vancouver, BC, Canada, Jul. 2007, pp. 3978-3982.

[16] A. Belatreche, L. P. Maguire, and T. M. McGinnity, "Advances in design and application of Spiking Neural Networks," *Soft Computing*, vol. 11, no. 3, pp. 239-248, 2007.

[17] G. Q. Bi and M. M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp.10464– 10472, 1998.

[18] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?," *Neural Computation*, vol. 17, no. 11, pp. 2337-2382, 2005.

[19] F. Ponulak, "ReSuMe - new supervised learning method for Spiking Neural Networks," Institute of Control and Information Engineering, Poznan University of Technology, Poland, Tech. Rep. 2005. [Online]. Available: http://d1.cie.put.poznan.pl/~fp/

[20] F. Ponulak, "Analysis of the ReSuMe learning process for spiking neural networks," *International Journal of Applied Mathematics and Computer Science*, vol. 18, no. 2, pp. 117-127, 2008.

[21] F. Ponulak and A. Kasinski, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification and spike-shifting," *Neural Computation*, vol. 22, no. 2, pp.467–510, 2010.

[22] R. Gutig and H. Sompolinsky, "The tempotron: a neuron that learns spike timing-based decisions," *Nature Neuroscience*, vol. 9, no. 3, pp. 420-428, 2006.

[23] R. V. Florian, "Tempotron-like learning with ReSuMe," in *Proceedings of the 18th International Conference on Artificial Neural Networks - ICANN 2008, Part II*, Prague, Czech Republic, Sep. 2008, pp. 368–375.

[24] T. Masquelier, R. Guyonneau, and S. Thorpe, "Spike timing dependent plasticity finds the Start of repeating patterns in continuous spike trains," *PLoS ONE*, vol. 3, no. 1, p. e1377, 2008.

[25] T. Masquelier, R. Guyonneau, and S. Thorpe, "Competitive STDP-based spike pattern learning," *Neural Computation*, vol. 21, no. 5, pp. 1259–1276, 2009.

[26] R. Legenstein., D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing dependent plasticity with application to biofeedback," *PLoS Computational Biology*, vol. 4, no. 10, p. e1000180, 2008.

[27] S. Scarpetta, F. Giacco, and A. de Candia, "Storage capacity of phase-coded patterns in sparse neural networks," *Europhysics Letters*, vol. 95, no. 2, 2011.

[28] S. Scarpetta and F. Giacco, "Associative memory of phase-coded spatiotemporal patterns in Leaky Integrate and Fire networks," *Journal of Computational Neuroscience*, vol. 34, no. 2, pp. 319–336, 2013.

[29] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, no. 6582, pp. 520-522, 1996.

[30] A. Delorme and S. Thorpe, "Face identification using one spike per neuron: Resistance to image degradation," *Neural Networks*, vol. 14, no. 6-7, pp. 795-804, 2001.

[31] A. Delorme, J. Gautrais, R.van Rullen, and S. Thorpe, "SpikeNet: A simulator for modeling large networks of Integrate and Fire neurons," *Neurocomputing*, vol. 26-27, pp. 989-996, 1999.

[32] A. Delorme, L. Perrinet, and S. Thorpe, "Networks of Integrate-and-Fire neurons using rank order coding B: Spike timing dependent plasticity and emergence of orientation selectivity," *Neurocomputing*, vol. 38-40, pp. 539-545, 2001.

[33] S. Gomes Wysoski, L. Benuskova, and N. Kasabov, "On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition," in *Proceedings of the 16th International Conference on Artificial Neural Networks*, Athens, Greece, Sep. 2006, pp. 61-70.

[34] K. Dhoble, N. Nuntalid, G. Indiveri, and N. Kasabov, "On-line spatiotemporal pattern recognition with evolving spiking neural networks utilising address event representation, rank order- and temporal spike learning," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, Brisbane, Australia, Jun. 2012, pp.554 – 560.

[35] F. Alnajjar, I. Bin Mohd Zin, and K. Murase, "A spiking neural network with dynamic memory for a real autonomous mobile robot in dynamic environment," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, Hong Kong, China, Jun. 2008, pp.2207 – 2213.

[36] A. Kasinski and F. Ponulak, "Comparison of supervised learning methods for spike time coding in spiking neural networks," *International Journal of Applied Mathematics and Computer Science*, vol. 16, no. 1, pp.101-113, 2006.

[37] J. Wang, A. Belatreche, L. P. Maguire, and T. M. McGinnity, "Online versus offline learning for spiking neural networks: A review and new strategies," in *Proceedings of the 8th IEEE International Conference on Cybernetic Intelligent Systems (CIS)*, University of Birmingham, Birmingham, UK, Sep. 2009, pp.1-6.

[38] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063-1070, Sep. 2004.

[39] S. Thorpe and J. Gaustrais, "Rank order coding," in *Computational Neuroscience: Trends in Research*, J. Bower, Ed. New York: Plenum Press, 1998.

[40] The AT&T dataset. [Online]. Available: http://www.cl.cam.ac.uk/Research/DTG/attarchive/facedatabase.html